Prof Michael Thielscher

Number Trails

Aim

A **number trail** is a series of $k \geq 1$ numbers:

$$\chi_1 \rightarrow \chi_2 \rightarrow \chi_3 \rightarrow \dots \rightarrow \chi_{k-1} \rightarrow \chi_k$$

in which

- numbers are in ascending order, and two consecutive
- numbers χ_i and χ_{i+1} differ by only one digit:
    1. either they are of the same length and are identical except in one position, e.g. `9024` → `9324`
    2. *or* one digit is added, e.g. `1234` → `12345` or `314` → `3014`.

An example is the following number trail of length $k = 8$:

`123` → `1234` → `1434` → `1444` → `10444` → `16444` → `76444` → `76544`

Your task is to develop a program to compute the *longest* number trails that can be built from a collection of numbers.

Input

Your program should start by prompting the user to input a positive number *n*, then prompt for *n* numbers.

Hint:

You may assume that

- the input is syntactically correct (a positive number $n \geq 1$ followed by *n* positive numbers); no
- number is larger than `INT_MAX` on a CSE computer (= 2147483647, defined in the standard library `limits.h`); there will be no more than 100 numbers (i.e., $n \leq 100$).
-

You can also assume that the number are input in ascending order (so you don't have to sort them yourself).

Output

- **Task 1:** For each number χ, compute and output all the numbers that could be used as the next number after χ in a trail.
- **Task 2:** Compute and output
    a. the length of the longest trail that can be built from the numbers in the input,
    b. all number trails of maximum length that can be built from the set of numbers in the input.

Stage 1

For stage 1, you need to demonstrate that you can build the underlying graph *under the assumption that all numbers from the input have the same number of digits.*

Any test for this stage will have only numbers of the same length and will be such that they all together form a trail. Hence, this stage focuses on Task 1, and for Task 2 you can just output *n* (= maximum length of a trail) and a trail containing all the numbers (= the only longest trail).

Here is an example to show the desired behaviour of your program for a stage 1 test:

```
prompt$ ./trails
Enter a number: 6
Enter a number: 9020
Enter a number: 9021
Enter a number: 9024
Enter a number: 9324
Enter a number: 9334
Enter a number: 9344

9020: 9021 9024
9021: 9024
9024: 9324
9324: 9334 9344
9334: 9344
9344:

Maximum trail length: 6

Longest trail(s):
9020 -> 9021 -> 9024 -> 9324 -> 9334 -> 9344
```

Stage 2

For stage 2, you should demonstrate that you can find *one* maximal trail under the assumption that all numbers have the same number of digits.

Any test for this stage will be such that all numbers are of equal length and there is only one longest number trail.

Here is an example to show the desired behaviour of your program for a stage 2 test:

```
prompt$ ./trails
Enter a number: 8
Enter a number: 214
Enter a number: 217
Enter a number: 414
Enter a number: 417
Enter a number: 514
Enter a number: 518
Enter a number: 614
Enter a number: 918

214: 217 414 514 614
217: 417
414: 417 514 614
417:
514: 518 614
518: 918
614:
918:

Maximum trail length: 5
Longest trail(s):
214 -> 414 -> 514 -> 518 -> 918
```

Stage 3

For stage 3, you should extend your program for stage 2 such that numbers can be of different magnitude, that is, with different numbers of digits.

Tests for this stage are also guaranteed to have just one number trail of maximum length.

Here is an example to show the desired behaviour of your program for a stage 3 test:

```
prompt$ ./trails
Enter a number: 6
Enter a number: 314
Enter a number: 3104
Enter a number: 3154
Enter a number: 5314
Enter a number: 5324
Enter a number: 53024

314: 3104 3154 5314
3104: 3154
3154:
5314: 5324
5324: 53024
53024:

Maximum trail length: 4
Longest trail(s):
314 -> 5314 -> 5324 -> 53024
```

Stage 4

For stage 4, you should extend your stage 3 program such that it outputs

- all number trails of maximal length in ascending order, that is, starting with the trail whose first
- number is the smallest of all trails. For trails that start with the same number, the trail whose second number is the smallest should come first, etc.

Here is an example to show the desired behaviour of your program for a stage 4 test:

```
prompt$ ./trails
Enter a number: 5
Enter a number: 314
Enter a number: 3104
Enter a number: 3154
Enter a number: 5314
Enter a number: 5324

314: 3104 3154 5314
3104: 3154
3154:
5314: 5324
5324:

Maximum trail length: 3
Longest trail(s):
314 -> 3104 -> 3154
314 -> 5314 -> 5324
```

Note:

- It is a requirement that the trails are output in ascending order.

Complexity Analysis

Your program should include a time complexity analysis for the worst-case asymptotic running time of your program, in Big-Oh notation, depending on the size of the input:

1. your implementation for Task 1, depending on the number *n* of numbers;
2. your implementation for Task 2, depending on the number *n* of numbers.

Your main program file `trails.c` should start with a comment: `/* … */` that contains the time complexity of your program in Big-Oh notation, together with a short explanation.